COBB

INSIDE DOS

Tips & techniques for MS-DOS & PC-DOS Versions 5 & 6

April 1994
Vol. 5 No. 4

VERSION 5.0 & 6.x

Using Doskey macros instead of batch files

Doskey macros were introduced in DOS Version 5. By then, many DOS power users were so comfortable using batch files that they overlooked the new macro capability. However, if you're concerned about conserving disk space, it may be time to reconsider converting some of your shorter batch files to macros. In this article, we'll give you an overview of Doskey macros: how they differ from batch files, their advantages and disadvantages, and how to create, run, edit, and delete a macro that finds files for you. We'll also give you a few tips on storing your Doskey macros.

What is a Doskey macro?

Doskey is a DOS terminate-and-stay-resident program. It stores in a buffer the commands you enter at the DOS prompt and lets you recall them to the command line. A Doskey macro, in essence, assigns a name to a command you enter, so you can recall the command later. Like the Doskey commands, DOS flushes Doskey macro definitions from the buffer when you reboot your computer.

For simplicity, think of Doskey macros as batch files that reside in your computer's memory. Like a DOS batch file, a Doskey macro allows you to store a command or a series of commands and execute them later by typing the macro's name at the DOS prompt.

Macros and batch files: A comparison

Macros and batch files have a lot in common, but there are differences, too; for example:

- Macros are limited to 127 characters on a single line, while batch files can be any length.
- You can run a batch file from another batch file or from a macro. You can define a macro in a batch file, but you can't run a macro from a batch file. Also, you can't run a macro from another macro.
- In the macro language, you represent operators and other symbols differently than you do in batch files. (We discuss the macro language in "Learning the Doskey Macro Language" on page 4.)
- You can run multiple commands in both, but in a macro, you use a special character instead of a carriage return to separate commands.
- Some batch file commands don't work in macros:
  ECHO OFF—all macro commands echo to the screen as they run.
  IF and GOTO—you can't use conditional logic in macros to branch to a subroutine.
- If DOS finds a macro with the same name as a DOS command or a batch file, the macro takes precedence. So, you can use a macro to redefine a DOS internal or external command.

One of the biggest advantages of Doskey macros over batch files involves how DOS stores them. DOS stores macros in RAM, so they take up a minimal amount of space. Batch files, on the other hand, consume a lot of disk space. When you save a batch file, DOS assigns it an entire allocation unit, or cluster. On most machines, a cluster is 2,048 bytes; on a compressed disk, the cluster is 8,192 bytes. So, even if your batch file is only 20 bytes, it will take up an entire cluster of disk space.

Another benefit of Doskey macros is that, because they're stored in memory, they generally execute faster than batch files. In addition, you can run a

IN THIS ISSUE

macro from any directory at any time. If a batch file isn't in the current directory or in a directory listed in your PATH statement, you must type a path name when you invoke it.

Of course, these benefits carry a price. Since a macro is stored in RAM—not on disk—you lose it when you reboot or turn off your computer. If you use a particular macro once in a blue moon, you won't mind losing it when you power down. However, you'll want to devise a storage scheme for macros you use regularly. Now let's look at how to create, use, edit, and delete a Doskey macro you create at the command line. Later, we'll look at how to define and store a macro within a batch file.

Creating and running macros

To create a Doskey macro, you simply enter

```
doskey macroname=command
```

at the DOS prompt, where *macroname* is the name you want to assign to your macro, and *command* is the complete command you want the macro to execute.

Once you've created a macro, you can execute its command or commands by simply typing its name at the DOS prompt

```
macroname
```

and pressing [Enter].

For example, you can create a macro to find and list a filename wherever it resides by entering this Doskey macro at any command prompt:

```
doskey whereis = dir $1 /s $b find /v "Volume"
```

Although this macro uses some unfamiliar symbols, its logic is fairly easy to follow. The DIR $1 /S command searches every directory and subdirectory on your system and lists a directory of the file or files you specified. The FIND command removes the volume label information from the display. Now if you want to locate a file, you simply type its name after the WHEREIS command you've just defined.

For example, to see the names and locations of all the batch files on your system, just enter

```
C:\>whereis *.bat
```

DOS will echo your command to the screen and then list the directory name and filename entry for each batch file it finds.

If you forget which macros you're currently running, you just issue the command

```
doskey /m
```

and DOS will list the macro names and the full macro statements.

INSIDE DOS

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group.

Prices: Domestic: $49/yr ($6.00 each)
Outside US: $69/yr ($8.50 each)

Phone: Toll free: (800) 223-8720
Local: (502) 491-1900
Customer Relations Fax: (502) 491-8050
Editorial Department Fax: (502) 491-4200

Address: You may address tips, special requests, and other correspondence to

The Editor, *Inside DOS*
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

For subscriptions, fulfillment questions, and requests for bulk orders, address your letters to

Customer Relations
9420 Bunsen Parkway, Suite 300
Louisville, KY 40220

Postmaster: Second class postage is pending in Louisville, KY. Send address changes to

Inside DOS
P.O. Box 35160
Louisville, KY 40232

Authorized Canada Post International Publications Mail (Canadian Distribution) Sales Agreement #XXXXXX CANADA GST #123669673. Send returns to Canadian Direct Mailing Sys. Ltd., 920 Mercer Street, Windsor, Ontario, N9A 7C2. Printed in the USA.

Staff:

Editor-in-Chief:	Janice Walter
Contributing Editor:	Van Wolverton
Editors:	Linda Recktenwald
	Cecilia Crosby-Lampkin
	Tessa Gavron
	Mary Jacobson
Production Artists:	Karen Collins
	Julie Jefferson
Managing Editor:	Mark Kimbell
Circulation Manager:	Marjorie Glassman
Editorial Director:	Jeff Yocom
Publishers:	Mark Crane
	Jon Pyles

Advertising: For information about advertising in Cobb Group journals, contact Tracee Bell Troutt at (800) 223-8720, ext. 430.

Back Issues: To order back issues, call Customer Relations at (800) 223-8720. Back issues cost $6.00 each, $8.50 outside the US. You can pay with MasterCard, VISA, Discover, or American Express, or we can bill you.

Advisory Board: Earl Berry Jr.
Tina Covington
Marvin D. Livingood

Editing Doskey macros

Once you test a macro, you may decide you want to change it. You can change a macro by retyping the entire command. However, the best way to edit a macro is to press ↑ until the Doskey buffer displays the macro definition on the command line. Then, since Doskey is loaded, you can press ← and → to place the cursor where you want and type the changes. You can press [Insert] to toggle Insert and Overstrike modes.

Let's say you want to add the /P switch to the WHEREIS macro so it will display only one screenful of information at a time. Simply press ↑ until the macro definition appears. Then press ← until the cursor is between */s* and *$b*. Now press the [Insert] key to enter Insert mode and type */p*. Press the [End] key to move the cursor to the end of the command and press [Enter] to enter the new definition of WHEREIS.

Deleting Doskey macros

There are two ways to delete macros you enter from the command line. You can delete all your current macros by rebooting or turning off your computer. You can selectively delete a macro by entering at the command line the DOSKEY command followed by the name of the macro and an equal sign. For example, you enter

```
doskey whereis =
```

to delete the WHEREIS macro.

Storage options for Doskey macros

As we said, one of the drawbacks of macros is that they disappear when you reboot or turn off your computer. Naturally, once you develop a nice library of Doskey macros, you won't want to spend hours each day retyping the ones you need. You'll definitely want to store your macros.

When it comes to storing macros, you have three basic choices:

- You can create the macros each time you boot your computer by placing your Doskey macros on individual lines in your AUTOEXEC.BAT file.
- You can create the macros each time you boot up by storing your macros in a batch file and calling that batch file from your AUTOEXEC.BAT file.
- You can place all your macros in a batch file and run the batch file only when you need to use one of your macros.

Of course, there are endless variations on these choices. To create a macro in your AUTOEXEC.BAT file, simply open the file in the DOS Editor or a compatible text processor and add the DOSKEY command the same way you'd enter it from the DOS prompt. You can edit or delete macros as you'd edit or delete any text you create in the Editor. If you don't want to clutter your AUTOEXEC.BAT file with a lot of macro commands, you might prefer to store your macros in a separate batch file.

To store macros in a batch file, create a batch file in a text editor, type each macro on its own line, and save the file with a name that includes the BAT extension. When you run the batch file, it will execute each line in turn, effectively entering Doskey macros into memory for you.

If you want to run the batch file from AUTOEXEC.BAT, place a line in your AUTOEXEC.BAT file that reads

```
call batchfile
```

You use the CALL command to execute a batch file from another batch file. CALL ensures that your system returns to your original batch file after it executes the second one.

You may even want to create several batch files for Doskey macros. For example, you might store your file management macros in FILEMACS.BAT, your memory management macros in MEMMACS.BAT, and so on. Then you can decide which set you want to invoke for the tasks at hand.

Conclusion

Doskey macros are a worthwhile alternative to batch files in many cases. In this article, we discussed the differences between batch files and macros and showed how to create, use, and store Doskey macros. ◆

VERSION 5.0 & 6.x

Learning the Doskey macro language

We showed you why you should consider converting some of your short batch files into Doskey macros in "Using Doskey Macros Instead of Batch Files" on page 1. The logic behind Doskey macros is the same as for batch files, but you use different characters to represent operators, variables, and other symbols. Table A shows the macro equivalents of common batch file and command line symbols.

If you haven't worked with macros very much, their syntax and appearance can seem foreign and intimidating. In this article, we'll discuss some ways to use Doskey macros and show how to use Doskey macro commands and symbols. Along the way, we'll share some of our favorite Doskey macros.

TABLE A: *Macro equivalents of batch file symbols*

These characters...	represent...
$G or $g	the output redirection symbol >
GG or gg	the append redirection symbol >>
$L or $l	the input redirection symbol <
$B or $b	the pipe operator |
$T or $t	the command separator (carriage return in batch files)
$1 through $9	the batch file parameters %1 through %9
$*	everything typed on the command line after the macro name
$$	the dollar sign

Using $t to stack commands

Do you use batch files to run applications? These short batch files typically are a few lines long and consist of only a few commands, yet each one takes up an entire sector of disk space. You can easily translate these batch files into Doskey macros. All you need is a simple tool: the command separator.

You can use $T or $t to emulate the carriage returns you'd use to separate commands in a batch file. Using the command separator, you can build some very complex macros—as long as you don't exceed the 127-character limit.

Let's look at the MSW macro. This macro changes to the Microsoft Works directory and starts the program. When you exit Works, the macro resumes control and changes to the root directory. Finally, MSW types a menu from which you can select another application. You place all the commands on a single line

```
doskey msw = cd\msworks $t works $t cd\ $t type menu.txt
```

simply by using the command separator, $t.

Allowing for variable information you enter after the macro name

Sometimes you want a macro to execute a series of commands. At other times you want the macro to act on information you enter after the macro name. If this is the case, you must tell DOS to act on that information by including in your macro definition a variable or parameter to hold that information.

Your macro might include commands that each act on a single piece of the information you add. If so, you'll need to assign each bit of information to a separate parameter. (We discuss replaceable parameters in the next section.)

On the other hand, if the commands in your macro act on your information as a group, you can probably use the more general placeholder, $*. The $* symbol holds all the text you type on the command line after the macro name. Common sense and experience will help you decide whether you need to use the variable symbol $* or specific parameters.

A simple application of the $* symbol involves creating Doskey macros for commands you frequently mistype. The following macros let you enter command switches, drive and directory designations, and filenames as you would with the original commands:

```
doskey copu = copy $*
doskey dor = dir $*
doskey ckdsk = chkdsk $*
```

You might think you can simply redefine a misspelled command name with the name of a legitimate command, as in *doskey copu=copy*. Well, you can't. DOS can't translate the misspelled command into the legitimate command until you press [Enter]—and after that, it's too late to add switches and filenames to the command. You must use $* to tell the legitimate command to act on the information you entered with the misspelled command.

Using replaceable parameters in your macros

In a batch file, you use percent signs to represent parameters (%1 through %9) that you enter after the name of a

batch file. In a macro, you represent these replaceable parameters using dollar signs ($1 through $9).

Let's look at the SWAP macro, which switches the names of two files. The macro uses three RENAME commands, two parameters, and a dummy file to do its work:

```
doskey swap = ren $1 xyz.### $t ren $2 $1 $t ren
     xyz.### $2
```

First, the macro gives a dummy name (*xyz.###*) to the first file ($1) you enter as a parameter. Next, it gives the second file ($2) the name of the first file. Finally, it renames the dummy file with the name of the second file. You invoke this macro by entering two filenames after the SWAP command.

Placing redirection symbols in macros

Perhaps the strangest looking characters in a Doskey macro are the redirection symbols. It's hard to think of the character $G as the output redirection symbol (>), but consider this: You're probably already familiar with the macro redirection characters because you can use the same characters ($L for <, $G for >, and $B for |) in a PROMPT statement. In the familiar *PROMPT=pg* statement, *$p* displays the current directory's path and *$g* displays >.

Below are a couple of Doskey macros that use redirection symbols. The first bypasses the PRINT command's *Name of list device [PRN]:* prompt by telling DOS the name of the printer up front. The macro

```
doskey print = echo lpt1 $b print $1
```

is the equivalent of the command

```
echo lpt1 | print filename
```

The second macro, SHELINFO, types the DOSSHELL.INI file one screen at a time by piping it through the MORE filter:

```
doskey shelinfo = more $l c:\dos\dosshell.ini
```

At the DOS prompt, you'd use the command

```
more < c:\dos\dosshell.ini
```

Using $$ to create prompt macros

Because Doskey macros use the dollar sign to define special characters, you must use two dollar signs in a macro command that includes a dollar sign. The macro below uses ANSI codes in a PROMPT command to blank the screen when you press the apostrophe followed by [Enter]. We use the characters *$el* to tell DOS that what follows is an ANSI code. In this macro, we add a second dollar sign to commands that need only one in a batch file:

```
doskey ' = cls $t prompt $$e[0;30;40m$$e[2J $t pause
     $t prompt $$e[37;40m$$p$$g
```

First, the macro clears the screen. Next, it issues ANSI codes that turn off the screen attributes, change the colors to black on black, and move the cursor to row 0. The now-invisible PAUSE command eliminates the cursor and waits for you to press a key. When you do, the second PROMPT command changes the prompt to white on black and displays the path and > character.

A special case: The FOR command

The FOR command has a slightly different syntax, depending on whether you're defining your macro at the command line or from a batch file. If you define a FOR macro at the DOS prompt, you use a single percent sign with the variable. If you define a FOR macro from a batch file, you use two percent signs.

The following macro, ATR, sets the attribute of whatever files you specify to read only. To define the macro at the DOS prompt, you enter

```
doskey atr = for %a in ($*) do attrib +r %a
```

To define the macro from a batch file, you enter

```
doskey atr = for %%a in ($*) do attrib +r %%a
```

To use the macro, you enter the macro name followed by the path and names of the files you want to make read only.

A note about length restriction

As we noted, macros are limited to 127 characters, of which the first seven are the word *doskey* and a space. For readability, you'll want to leave spaces around each command in your macro.

If your macro pushes the 127-character limit, however, you can choose a shorter name for the macro, or you can remove the spaces before and after the equal sign and the command separator ($t). Of course, if that doesn't work, you may need to resort to creating a batch file.

Conclusion

You can accomplish many of the same tasks using Doskey macros as you can using a short batch file. This article showed you how to create a variety of macros using Doskey macro commands and symbols. ◆

VAN WOLVERTON

VERSION 6.x

The hot question: Should I use file compression?

One of the benefits of living in a rural area while earning your living writing about computers is being treated as the fount of all computer knowledge. Whether or not you know the answer, you're asked all the questions that come along.

The questions aren't always the same, of course. Years ago, one of the most common questions I heard was "Should I get a hard disk?" (They were a lot more expensive a decade or so ago.)

That question has given way to "What's Windows all about? Do I need it?" and another generation of questions: "Should I replace this 286 machine I bought because you recommended it back at the dawn of time?" (The implied accusation is best left unanswered, especially if the questioner has already discovered that she'll be lucky to get $500 for the machine that you suggested she pay $2,000 for six years ago.)

Those, of course, are easy questions. Too often, the hard ones crop up in the Top Ten list, and sometimes they reach the top. That's the case right now. The question I'm asked more than any other doesn't have an easy answer: "Should I use DoubleSpace to compress my files?" As with so many tough questions, the best answer is pretty unsatisfactory: "Well, it depends..."

DOS still won't lose your data

I wrote a few months ago that your files were safe with DOS 6, despite a host of horror stories in letters to the editor and online forums about files and even disks irretrievably damaged by DoubleSpace, the file-compression program bundled with DOS 6. I wasn't able to turn up any evidence of reproducible problems. More important, neither were the major software testing installations such as Ziff-Davis Labs. Nothing has been discovered since then to change that advice significantly, although I might counsel caution in some circumstances. More on that in a moment.

To quell the doomsayers, Microsoft came out with Version 6.2 just six months after 6.0. It includes several safety features that make it even less likely that you'll have problems with DoubleSpace. These additional safety features notwithstanding, Microsoft was mostly fixing an image problem, not a DOS problem.

File compression isn't risk-free

The technique of compressing a file is neither new nor mysterious. Following principles discovered and put into practice by code makers (and code breakers) centuries ago, you can shrink a file by using short codes to represent longer words or phrases that appear frequently in the file. For example, if you knew that a file wouldn't contain a dollar sign ($), you could save quite a bit of space by replacing each occurrence of the word *the* with a dollar sign. Depending on the source of the document, you could save even more space by letting the dollar sign represent longer repetitive sequences such as *local area network* or *cultural diversity*.

Compressing a file in this fashion, of course, requires that a program read the contents of the file and make the required substitutions. Before the file can be used again, another program must expand the file by reading its contents and undoing all the substitutions. Programs that do this on an individual file basis have been used for years to reduce the length of files transmitted over telecommunications lines (saving time and money) and to reduce the amount of disk space needed to store infrequently used files.

All this is not to say that file compression is without risk. Compression and expansion programs can have errors, or something can go wrong (such as a power surge or sag) while compression or expansion is occurring, causing incorrect data to be written in the disk file. Because file-compression programs change the content of a file drastically, it can be difficult to reconstruct a compressed file if it's been changed even slightly.

The danger of data damage is slight when you're using a standalone file-compression program, primarily because these programs don't require any special treatment from DOS: Like many other programs, they simply read a file, do some processing, and store a revised version of the file. They're also used relatively infrequently, reducing the window of vulnerability to external problems such as power failures.

But file-compression programs such as DoubleSpace or Stacker that run all the time inevitably create a higher risk. They *do* require special treatment from the operating system, because each time DOS asks for a file, the compression program intercepts the request that normally would go to the disk controller. Then the compression program either (1) compresses the data in memory and writes it on the disk in order to store a file, or (2) reads the compressed file from disk, expands it, and makes the restored information available to DOS in order to open a file. It does this for every single file DOS uses, dozens or hundreds of times in a single session, leaving the window of vulnerability open much longer.

In addition, all the compressed files—although to you they seem like individual files—are, in fact, stored in one huge, hidden file that the file-compression program treats as if it were a disk drive. A hard disk problem that affects a portion of that enormous file could cost you *all* your files.

But we need bigger disks

So if our data is valuable—and whose isn't—and there's even the slightest risk of losing it, why even consider file compression? Because we're running out of disk space, that's why.

When IBM announced the PC-XT in 1983, it made headlines because it had a whopping 10Mb hard disk, the first widely available personal computer that came with such commodious storage. When the PC-AT came along a few years later, it knocked everybody over with its 30Mb hard disk. Who could ever need such vast reaches of disk storage?

Times have changed, with a vengeance. DOS 6.2 takes more than 8Mb of disk space. Windows requires another 15Mb or so. Many Windows applications, such as Corel Draw or Word for Windows, take up 30Mb or more. A full installation of Microsoft Office—the bundle (or *suite*, as they prefer) that includes Word, Excel, Power Point, and Access—gobbles up more than 65Mb! Add DOS and Windows, both of which you must have in order to run Office, and you've used up almost 90Mb on your disk. And, you haven't yet stored a single data file.

So what should you do?

Well, it depends. If you have a small hard disk—meaning less than 100Mb—then yes, by all means, use file compression to increase your disk storage as much as possible. You're bound to run short unless you don't use Windows, you use only a few application programs, and you don't keep many data files hanging around.

If you have a large hard disk—meaning 320Mb or more—I wouldn't recommend bothering with file compression unless your work involves graphics or desktop publishing, thereby loading up your hard disk with huge graphics files and jillions of fonts. If you don't use those space-grabbing programs, chances are you can keep things under control with occasional housekeeping sessions to cull the files you no longer need (something you should do anyway).

If you use Windows, you probably should use file compression. Chances are you'll need the space unless you've got a really big hard disk. File compression exacts a small performance penalty, and Windows already is noticeably slower than DOS, but that's the price you have to pay for sufficient disk storage.

If you don't back up your hard disk regularly, don't use disk compression. You're already living dangerously; it makes no sense to increase the risk, even minimally.

If you already use PKZip, LHArc, or some other standalone file-compression program to compress selected files, don't bother with full-time file compression. You've probably compressed the biggest data files, and the most you could gain would be the compression of program files. If you do decide to use full-time data compression, stop compressing individual files, because there's nothing to be gained by compressing twice. In fact, you should expand the files you've already compressed, because compressing a compressed file takes more time than compressing an uncompressed file, and the result often takes up more space than a file that is compressed just once.

For most of you, the answer is probably the same as it was six months ago: Yes, go ahead and use DoubleSpace (or the full-time compression program of your choice). But protect yourself by running ScanDisk weekly to check for problems in your file structure and disk surface, and use Defrag to pack together the files on your disk. You'll maintain optimum performance and catch problems before they get severe enough to threaten your files. ◆

DOS FILE MAINTENANCE

VERSION 6.x

Quickly deleting hidden, system, and read-only files

DOS doesn't let you use the DELETE or ERASE command to delete files that carry one or more of the hidden, system, and read-only attributes. To delete such files from the command line in Version 5, you must first use the ATTRIB command to remove the attributes from the file(s).

Fortunately, DOS Version 6.0 provides a quick way to delete files that carry hidden, system, and read-only attributes. You simply use the DELTREE command followed by the name of the file you want to remove. However, if you delete a file using DELTREE, it can't be recovered using UNDELETE. Use DELTREE only on files you want to remove permanently. Now, let's look at how you can use DELTREE to remove individual files and groups of files.

Using DELTREE to delete files

To delete files using DELTREE, you just enter the command

```
deltree filespec
```

where *filespec* is the file specification (complete path and filename) of the file you want to delete.

You can type multiple filenames after the command—as long as you stay within the 127-character limit on commands you enter from the DOS prompt. To delete multiple files, you enter the command followed by the file specifications:

```
deltree filespec1 filespec2 filespec3... filespecn
```

Be sure to type a space between each file specification.

When you enter the DELTREE command, DOS gives you a chance to change your mind. DOS displays the prompt

```
Delete file "filespec"? [yn]
```

for each file you specified. You can cancel the operation by pressing *N* and then [Enter]; or, you can press *Y* and then [Enter] to permanently remove the file.

While DOS lets you use wildcard characters with DELTREE to delete groups of files, you should do so carefully. When you use wildcards, DOS often finds files you don't intend to act on. Even though DOS prompts you to verify each filename you specify, it's all too easy to inadvertently press the wrong key. Now let's delete some files.

An example

Let's suppose you keep information on a diskette in two directories. You keep business information in visible files and confidential information in invisible files. To make the files invisible, you've attached the hidden attribute to them. When you run a DIR command on one of the directories, you see this information:

```
Directory of B:\HAMLET

.           <DIR>          01-03-93      4:17p
..          <DIR>          01-03-93      4:17p
STAGING           2,036    04-19-93      5:12p
ADVERTIS          3,243    04-26-93      9:12a
SETS              4,102    04-26-93     12:34p
        5 file(s)          9,381 bytes
                         692,200 bytes free
```

Now you want to delete all the confidential files. First you need a list of all the visible and invisible files in the directories and subdirectories on the diskette. To display the list, enter the command

```
C:\>attrib b: /s
```

When you do, you see this listing:

```
A     HR    B:\MUCHADO\BENEDICK
A     HR    B:\MUCHADO\CLAUDIO
A     H     B:\MUCHADO\DOGBERRY
A     H     B:\MUCHADO\HERO
A     HR    B:\MUCHADO\BEATRICE
A           B:\MUCHADO\SCRIPTS
A           B:\MUCHADO\SETS
A           B:\MUCHADO\ADVERTIS
A           B:\HAMLET\STAGING
A     HR    B:\HAMLET\OPHELIA
A     HR    B:\HAMLET\CLAUDIUS
A     H     B:\HAMLET\HAMLET
A     HR    B:\HAMLET\GERTRUDE
A           B:\HAMLET\ADVERTIS
A           B:\HAMLET\SETS
```

Now you can start deleting individual files or several files at a time. For example, enter the command

```
C:\>deltree b:\hamlet\claudius
```

at the prompt, and DOS asks you to verify your choice:

```
Delete file "b:\hamlet\claudius"? [yn]
```

As soon as you press *Y* and then [Enter], DOS permanently removes the file.

To delete several files, just enter the DELTREE command followed by the file specifications of all the files you want to delete (or as many as you can specify in 127 characters or less). For example, you could change to the B drive and enter this much of the command

```
B:\>deltree muchado\benedick muchado\claudio
    muchado\dogberry muchado\hero muchado\beatrice
    hamlet\ophelia hamlet\hamlet hamlet\ge
```

before you exceed 127 characters. But that's a lot of typing.

As an alternative, let's change to one of the directories and delete the appropriate files. To delete files in B:\MUCHADO, change to that directory and type the command

```
B:\MUCHADO>deltree benedick claudio dogberry hero beatrice
```

When you do, DOS prompts

```
Delete file "benedick"? [yn]
```

Press *Y* and then [Enter]. DOS displays the message

```
Deleting benedick...
```

and immediately prompts

```
Delete file "claudio"? [yn]
```

As soon as you press *Y* and then [Enter], you see the next message, followed by the next prompt. This continues until all the files you specified are taken care of. You can simply repeat the procedure to delete all the appropriate files in the B:\HAMLET directory. ◆

VERSION 6.x

Extending your path beyond the 122-character limit

By Greg Shultz

As you may know, the PATH command allows you to specify a list of directories that contain the executable files you want to be able to access from anywhere on your hard disk. However, because the PATH command is a *standard* DOS command, it has one major drawback—it's limited to 127 characters. Once you subtract the four characters for the command name and one for the space that comes between the command and the directory list, you have only 122 characters for listing the names of the directories you want in your actual path. If you have many applications on a large hard disk, use long names for your directories, or have executable files nested in directories several layers deep, you know how easy it is to exceed the 122-character limit.

To work around this limitation, you can restrict the number of directories you include in your actual path, or you can rename all your directories with two- or three-character names in order to cram as many directory names in the path as you can. However, using either of these methods can be a tremendous hassle. Fortunately, DOS 6 allows you to bypass the DOS command-line limit by specifying your path in the CONFIG.SYS file. When you do, there's no limit to PATH's length.

In this article, we'll explain why it's possible in DOS 6 to specify a path in the CONFIG.SYS file. Then, we'll show you how to set up a PATH command in your CONFIG.SYS file.

CONFIG.SYS and DOS 6

When Microsoft's programmers created DOS 6, they expanded the number of CONFIG.SYS directives (commands) from the 15 found in DOS 5 to 17. The two new directives are NUMLOCK and SET. Like the SET command, the SET directive functions allow you to specify a path from within the CONFIG.SYS file.

The SET command

As you may know, you can usually find the SET command in the AUTOEXEC.BAT file, where the command assigns a text string to an environment variable. Once assigned, an environment variable exists in memory and can be referenced by DOS or an application at any time. For example, many Windows applications use temporary files for storing data while they're running. These applications usually look for an environment variable called TEMP, which specifies the directory where the temporary files go. The command

```
SET TEMP=C:\WINDOWS\TEMP
```

in your AUTOEXEC.BAT file sets up the TEMP environment variable.

You can see a list of your system environment variables at any time by simply typing

```
SET
```

at the DOS prompt and pressing [Enter]. When you do, you'll see a list of environment variables followed by equal signs (=) and the text strings assigned to them. One of the environment variables you'll see is PATH. For example, your PATH environment variable might look similar to

```
PATH=C:\;C:\DOS;C:\WINDOWS;C:\CPS;C:\WINWORD;
     C:\EXCEL;C:\PDOXWIN
```

As you might expect, the command

```
PATH C:\;C:\DOS;C:\WINDOWS;C:\CPS;C:\WINWORD;
     C:\EXCEL;C:\PDOXWIN
```

in your AUTOEXEC.BAT file is simply a shorthand notation for

```
SET PATH=C:\;C:\DOS;C:\WINDOWS;C:\CPS;C:\WINWORD;
     C:\EXCEL;C:\PDOXWIN
```

The PATH command in the AUTOEXEC.BAT file simply establishes an environment variable in memory exactly as the SET command does. Since the SET command is now available from CONFIG.SYS, you can easily set up the path in the CONFIG.SYS file.

Overcoming the 122-character limit

As we mentioned, DOS limits commands to 127 characters. Once you subtract the five characters consumed by the environment variable name (PATH) and the space or equal sign that follows it, you have only 122 characters for the list of directories that make up the path itself. However, because CONFIG.SYS directives aren't really DOS commands and the SET command is now also a CONFIG.SYS directive, there's no limit to the length of a PATH string established in the CONFIG.SYS file.

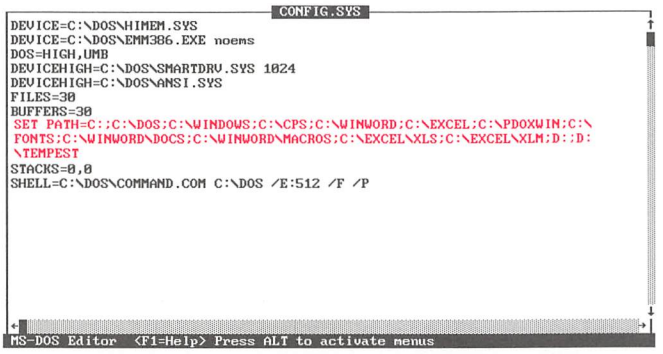
Making the change

Now that you see how you can use CONFIG.SYS to overcome the 122-character limit DOS imposes on the PATH command, making the change is easy.

You can do so quickly by cutting and pasting with the DOS Editor.

To begin, type *edit autoexec.bat* to open the AUTOEXEC.BAT file in the DOS Editor and then highlight the line containing the PATH command. Next, press [Shift][Delete] to cut the line to the Clipboard. Then, save the AUTOEXEC.BAT file.

Figure A



Placing the PATH command in your CONFIG.SYS is a snap.

Now, open your CONFIG.SYS file, position the cursor on any line, and press [Enter] to create a blank line. Type *SET* on the blank line and press [Spacebar]. Next, press [Shift][Insert] to paste the PATH command into your CONFIG.SYS file. Then, place an equal sign between the PATH environment variable and the list of directories, as shown in Figure A.

At this point, you can add to your path any of the directories that contain executable files you want to be able to access from anywhere on your hard disk. When you've finished adding directories, save your CONFIG.SYS file, close the Editor, and reboot your system. When you do, your new, longer path takes effect.

Conclusion

If you use DOS 6 and feel restricted by the PATH command's 122-character limit, you can bypass this limitation by using the new SET directive to place the path in your CONFIG.SYS file. In this article, we've explained how this technique works, and we've shown you how to set up a PATH command in your CONFIG.SYS file. ◆

Greg Shultz is editor-in-chief of Inside PC Tools for Windows *and* Inside Norton Desktop for Windows, *two other Cobb Group publications.*

DOS FILE MAINTENANCE

VERSION 5.0 & 6.x

Deleting old versions of DOS

In little more than a year, Microsoft has upgraded DOS twice. If you've installed even one of the new versions, you've undoubtedly noticed one or more OLD_DOS.# directories on your system. The C:\OLD_DOS.1 directory contains your most recent version, C:\OLD_DOS.2 contains the version before that, and so forth.

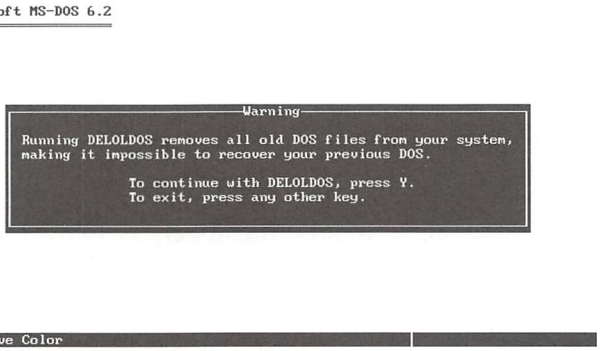
Since these directories can take up lots of space, you probably wonder if it's safe to delete them. If you're satisfied that all your applications work properly with your new version of DOS, and if you're absolutely sure you won't want to return to your old DOS version, the answer is a resounding *Yes*.

DOS even provides a command, DELOLDOS, just for the purpose of deleting old versions. All you have to do to remove old DOS directories is enter the command

```
deloldos
```

at the prompt. When you do, DOS displays the screen shown in Figure A, which asks you to confirm that you want to remove all of your old DOS directories. Just press *Y*, and DOS will remove all traces of every OLD_DOS.# directory from the root directory. DOS will also remove the DELOLDOS command from the C:\DOS directory.

Figure A



DELOLDOS asks if you want to remove all OLD_DOS.# directories.

What if your old DOS files are in a different directory?

If you renamed your previous DOS directory before you installed the most recent version, DELOLDOS won't be able to delete your old DOS files. If you renamed your directory, DOS indeed created an OLD_DOS.1 directory, but it contains only a README.NOW file. Running DELOLDOS removes only the OLD_DOS.1 directory and its README.NOW file. The files in your old DOS directory remain intact.

Also, if you changed the name of your OLD_DOS.# directory after you installed the new version, DELOLDOS will be unable to find it and delete it. In either of these situations, you'll have to manually delete your old DOS directory.

In Version 6.x, you can use the DELTREE command followed by the name of the directory you wish to delete. In Version 5, you'll have to use the DEL command to remove the files and the RD command to eliminate the directory; or, you can delete files and directories from the DOS Shell. ◆

LETTERS

Creating a safety net for your batch files

You sometimes criticize DOS because it didn't build safety into many of its old commands, but then you publish NUKE.BAT, a potentially lethal batch file, without even an "Are you sure you want to delete %1?" This would be a good place to use the new DOS CHOICE command, don't you think?

Doug MacGregor
Brookline, Massachusetts

Mr. MacGregor is absolutely right. We should have built in a safety net for NUKE.BAT, and DOS 6's CHOICE command is the perfect solution. We introduced NUKE.BAT in the January 1994 article "Using NUKE.BAT to Permanently Delete Files," and warned you to use it carefully. Naturally, you can't always pass along such warnings to each of *your* users!

We modified our copy of NUKE.BAT to include a verification prompt. Our verification prompt comprises the VERIFY routine and the CANCEL routine. You can add to your copy the VERIFY and CANCEL routines shown in Figure A. You can adapt this safety feature for any batch file, so we've included in Figure A only a "bare bones" version of the original batch file.

The VERIFY routine warns that you're about to delete files and uses a DIR command to list the file or

Figure A

```
@echo off
rem NUKE_6.BAT revises Jan. 94 NUKE.BAT for 6.x users

if exist %1 goto VERIFY
goto OOPS

:VERIFY
cls
echo You're about to permanently remove the
echo following file(s):
echo.
dir %1 /b
echo.
echo Press Y to permanently delete the above
echo file(s). Press N to cancel. If you don't
echo choose in 10 seconds, NUKE_6.BAT will quit
choice without deleting any files. /t:n,10 /n
if errorlevel 2 goto CANCEL
if errorlevel 1 goto CANCEL

:KILL
rem KILL ROUTINE GOES HERE.

:CANCEL
echo NUKE_6.BAT cancelled! No files were deleted.
goto END

:OOPS
rem ERROR MESSAGE GOES HERE.

:END
```

The VERIFY and CANCEL routines let you cancel NUKE_6.BAT without permanently removing any files.

files you've specified. The /B switch specifies *bare* form, which retrieves only the filename(s). So, if you specified a path to the file(s) you want to delete, this DIR command won't list the name of the directory the files are in.

The CHOICE command includes the switch /T;N,10. This switch acts as a timer, waiting a specified time before it processes the response you specify. In this case, CHOICE waits ten seconds before it processes the N (for no) response. The command also includes the /N switch, which tells CHOICE not to display the valid choices in the prompt. We didn't specify the valid choices, so the command uses the default choices, YN (for yes and no, respectively).

CHOICE works by assigning an ERRORLEVEL to each key you offer the user. In our command, CHOICE assigns the default choices, Y and N, to the ERRORLEVELs 0 (zero) and 1, respectively. CHOICE assigns the ERRORLEVEL 2 to the third choice, which lets ten seconds pass without pressing either Y or N.

Once you know the ERRORLEVEL assignments, you can use conditional statements to send control to the appropriate routine. In plain English, the statement *if errorlevel 2* means *if you let ten seconds go by without pressing Y or N* and the statement *if errorlevel 1* means *if you pressed N*. Our two IF ERRORLEVEL statements send control to the CANCEL routine, which simply verifies that you've cancelled out of the program.

We didn't include an IF ERRORLEVEL statement for the Y response because we don't need it. If neither of the IF ERRORLEVEL statements sends control to the CANCEL routine, control will automatically pass to the command that follows. In this case, that's the first command in the KILL routine, which is what you'd specify in an *if errorlevel 0* statement anyway.

What about DOS 5 users?

DOS 5 doesn't offer the CHOICE command, and there's no simple way to create one using DOS commands. We'll explore creating a DOS 5 version of the CHOICE command at another time. However, there's a quick-and-dirty method for including a verification prompt in your batch files.

We show the NUKE_5.BAT file in Figure B. We added only a VERIFY routine to this bare bones version of NUKE.BAT.

As you can see, NUKE_5.BAT's VERIFY routine prompts you to press [Ctrl]C to exit the batch file. (Of course, you can use the [Ctrl]C keypress to cancel *any* batch file.) It's a simple and effective solution, but it has one drawback: After you press [Ctrl]C, DOS displays the somewhat cryptic message

```
Terminate batch job (Y/N)?
```

If your user doesn't know what this prompt means, he or she might inadvertently press *N* and the batch file will pick up with the KILL routine. The file(s) will be gone forever. This method is a less-than-perfect solution, but it offers a lot more security than omitting a verification prompt altogether. ◆

Figure B

```
@echo off
rem NUKE_5.BAT revises Jan. 94 NUKE.BAT for 5.0 users

if exist %1 goto VERIFY
goto OOPS

:VERIFY
cls
echo You're about to permanently remove the
echo following file(s):
echo.
dir %1 /b
echo.
echo Press [Ctrl]C to cancel the command.
echo To proceed, press any other key...
pause > nul

:KILL
rem KILL ROUTINE GOES HERE

:OOPS
rem OOPS ROUTINE GOES HERE

:END
```

NUKE_5.BAT lets you use the common [Ctrl]C keystroke to cancel out of the batch file.

Printed in the USA
This journal is printed on recycled and recyclable paper.
50% recycled fiber including 10% post-consumer waste paper